Udacity Data Wrangling Project Report

Introduction:

The Place chosen by me for this project is Ottawa, Canada. I chose this place because my brother lives there and I have heard lots of great stuff about that place. I also will be moving there in few years, so why not get to know about that city. The uncompressed file is around 920mb.

Link : Ottawa, Canada

Problems in the Map:

The map was analyzed properly was various problems using various techniques. I have decided to clean the map in the following areas:

- Spell out Numbers in the "Line" until 10
- Spell out the Street type & Directions
- Convert Province name to Province Code
- Convert phone numbers to international format with spaces
- Format Postal code to upper case & Discard the incorrect ones

Now let's see how each of these problems have been dealt with:

1. Spell out Numbers in the "Line" until 10

Here, the name "5th Line" is converted to "Fifth Line"

```
num_line_street_re = re.compile(r'\d0?(st|nd|rd|th|)\s(Line)$', re.IGNORECASE)

num_line_mapping = {
                     "1st": "First",
                     "2nd": "Second",
                     "3rd": "Third",
                     "4th": "First",
                     "5th": "Second",
                     "6th": "Third",
                     "7th": "First",
                     "8th": "First",
                     "9th": "Ninth",
                     "10th": "Tenth"
                   }

if num_line_street_re.match(name):
    nth = nth_re.search(name)
    name = num_line_mapping[nth.group(0)] + " Line"
    return name
```

2. Spell out the Street type & Directions

Here, the street names & directions like "Main St N" is converted to "Main Street North"

```
nth_re = re.compile(r'\d\d?(st|nd|rd|th|)', re.IGNORECASE)
nesw_re = re.compile(r'\s(North|East|South|West)$')

mapping = {
            "St": "Street",
            "St.": "Street",
            "Ave": "Avenue",
            "Ave.": "Avenue",
            ...
            "S.": "South",
            "S": "South",
            "W.": "West",
            "W": "West"
          }

street_mapping = {
                                    # same as above, minus North, East, South, West
                                  }
...

else:
    original_name = name
    for key in mapping.keys():
        # Only replace when mapping key match (e.g. "St.") is found at end of name
        type_fix_name = re.sub(r'\s' + re.escape(key) + r'$', ' ' + mapping[key],
original_name)
        nesw = nesw_re.search(type_fix_name)
        if nesw is not None:
            for key in street_mapping.keys():
                # Do not update correct names like St. Clair Avenue West
                dir_fix_name = re.sub(r'\s' + re.escape(key) +
re.escape(nesw.group(0)), \
                                                         " " + street_mapping[key] +
nesw.group(0), type_fix_name)
                if dir_fix_name != type_fix_name:
                    return dir_fix_name
        if type_fix_name != original_name:
            return type_fix_name
return original_name
```

3. Convert Province name to Province Code

Conversion of full province name to province code.

- "Ontario" to "ON"

```
# Change Ontario to ON
if province == 'Ontario':
    province = 'ON'
```

4. Convert phone numbers to international format with spaces

Conversion of numbers to international format :"+1 ### ### ####".

- "416-987-1234" to "+1 416 987 1234"

```
PHONENUM = re.compile(r'\+1\s\d{3}\s\d{3}\s\d{4}')

def update_phone_num(phone_num):
    # Check for valid phone number format
    m = PHONENUM.match(phone_num)
    if m is None:
        # Convert all dashes to spaces
        if "-" in phone_num:
            phone_num = re.sub("-", " ", phone_num)
        # Remove all brackets
        if "(" in phone_num or ")" in phone_num:
            phone_num = re.sub("[()]", "", phone_num)
        # Space out 10 straight numbers
        if re.match(r'\d{10}', phone_num) is not None:
            phone_num = phone_num[:3] + " " + phone_num[3:6] + " " + phone_num[6:]
        # Space out 11 straight numbers
        elif re.match(r'\d{11}', phone_num) is not None:
            phone_num = phone_num[:1] + " " + phone_num[1:4] + " " + phone_num[4:7] \
                        + " " + phone_num[7:]
        # Add full country code
        if re.match(r'\d{3}\s\d{3}\s\d{4}', phone_num) is not None:
            phone_num = "+1 " + phone_num
        # Add + in country code
        elif re.match(r'1\s\d{3}\s\d{3}\s\d{4}', phone_num) is not None:
            phone_num = "+" + phone_num
        # Ignore tag if no area code and local number (<10 digits)
        elif sum(c.isdigit() for c in phone_num) < 10:
            return None
    return phone_num
```

5. Format Postal code to upper case & Discard the incorrect ones

Discard postal codes that are not in the correct Canadian format, i.e., A1A 1A1, where A is a capital letter and 1 is an integer.

- "M36 0H7"

```
POSTCODE = re.compile(r'[A-z]\d[A-z]\s?\d[A-z]\d')

m = POSTCODE.match(post_code)
    if m is not None:
        # Add space in middle if there is none
        if " " not in post_code:
            post_code = post_code[:3] + " " + post_code[3:]
        # Convert to upper case
        new['value'] = post_code.upper()
    else:
        # Keep zip code revealed in postal code audit for document deletion purposes
        if post_code[:5] == "14174":
            new['value'] = post_code
        # Ignore tag if improper postal code format
        else:
            return None
```

So these are the few cleaning & auditiong process carried out on the data set.

Overview of the Data

The Original OSM XML chosen is more than 50mb when uncompressed. Here are a few statics of the dataset:

Size of the file : 950mb
Size of the Sample : 38mb

The number of nodes taken into the sample file is 25. The size of the sample taken is approximately 4% of the original file

- Number of unique Users : 1250

```
sqlite> SELECT COUNT(DISTINCT(e.uid))
                FROM (SELECT uid FROM Nodes UNION ALL SELECT uid FROM Ways) e;
```

- Number of Nodes : 3765450

```
sqlite> SELECT COUNT(*) FROM Nodes;
```

- Number of Ways : 494588

```
sqlite> SELECT COUNT(*) FROM Ways;
```

Additional Statistics:

- Number of Amenities: 11,798

```
sqlite> SELECT COUNT(*) FROM nodesTags WHERE WHERE nodesTags.key='Amenity'
```

- Number of Town halls: 24

```
sqlite> SELECT COUNT(*) FROM nodesTags WHERE WHERE nodesTags.key='townhall'
```

- Number of Libraries:

```
sqlite> SELECT COUNT(*) FROM nodesTags WHERE WHERE nodesTags.key='library'
```

- Most popular Fast Food Joints :

Subway	19
McDonalds	17
Dominoes Pizza	8
Wendy's	5
Centertown Donair & Pizza	4

```
sqlite> SELECT nodesTags.value, COUNT(*) as num
        FROM nodesTags
            JOIN (SELECT DISTINCT(id) FROM nodesTags WHERE value='fast_food') i
            ON nodesTags.id=i.id
        WHERE nodesTags.key='name'
        GROUP BY nodesTags.value
        ORDER BY num DESC
        LIMIT 5;
```

- Most Popular Banks:

TD Canada Trust	124
Scotiabank	78
CIBC	63
BMO Bank of Montreal	45
RBC	32

```
sqlite> SELECT nodesTags.value, COUNT(*) as num
        FROM nodesTags
            JOIN (SELECT DISTINCT(id) FROM nodesTags WHERE value='bank') i
            ON nodesTags.id=i.id
        WHERE nodesTags.key='name'
        GROUP BY nodesTags.value
        ORDER BY num DESC
        LIMIT 5;
```

Other ideas about the dataset

There are so many fields in the dataset, which are empty. The ratio to which the number of nodes available and the information for that particular field available is excessively low to analyze and make predictions. Few examples are “Blindaccessible”, “Wheelchair”, ”freeway”, etc.

Number of Nodes: 3765450

```
sqlite> SELECT COUNT(*) FROM Nodes;
```

Number of nodes with blind accessibility information: 1522

```
sqlite> SELECT COUNT(*) FROM nodesTags WHERE key='blind';
```

Percentage of nodes with blind people accessibility information:

1522/ 3765450= **0.00040 %**

With this extensive low information, it is very difficult to show informatics on blind people accessible places. So this can be improved in this dataset by adding more information to the all public places, which will be really useful.

And also, so many innovations can be made with this available data. These statistics can be used during the election to see which party has done the most for the welfare of the people in that city. With these statistics, the living of the people can be vastly improved. Things like number of fire stations, hospitals, pharmacies, etc. available per area can be calculated and can be checked if it meets the required number on a ratio based system.

This problem can be overcome by motivating people to contribute to the openstreetmap data. People should add in their local neighborhood places whichever is aware by them so that it helps in a larger & a useful database. A simple way would be to asking people to check-in at the places with their smartphones where ever they visit, so that with the help of their gps location, the places will be added to the database.

Benefits:

- A Larger and a much useful database
- Accurate statistics

Anticipated Problems:

- When more end users contribute to the database, it might tend to become a messy database
- A lot of cleaning & auditing process have to be done in order to keep the database clean and informative

Conclusion

Thus the Ottawa Streetmap dataset was analyzed, cleaned & processed into a csv file with the help of all the python scripts and data wrangling techniques learnt from data wrangling course & problem sets. This is very useful when a data set is very messy and to extract useful information from it.